Makani Final Thoughts

Project WE-201904

Final Memo
April, 2020

PREPARED FOR:

Makani Technologies, LLC

2175 Monarch St.
Alameda, California 94501
United States

PREPARED BY:

Rick Damiani, Ph.D., P.E.

rdamiani@RRDEngineering.com

Tel: +1-970-581-8091

RRD ENGINEERING
18960 W 92nd Dr.

Arvada, Colorado 80007, USA

EXECUTIVE SUMMARY

Makani Technologies, LLC (Makani) funded the development of an aero-hydro-servo-elastic computational tool (KiteFAST) to simulate the dynamics and internal loads of its airborne wind energy (AWE) kites.

RRD Engineering, LLC (RRD) supported the troubleshooting and the development of the software and conducted a verification and validation campaign of the results.

The efforts have been extensively documented in a Google Doc "ThoughtsOnController.docx", and in jupyter notebooks, which can also be interpreted as journals showing the steps taken and the main results of the verification and validation campaign. CSIM (Makani's simulation and control software) and empirical data from test flights were all used as benchmarks for the campaign and to improve KiteFAST. This memo summarizes the work conducted on the development of KiteFAST, lessons learned, and suggested path forward.

New capabilities were added to KiteFAST, especially with regard to the handling of the controller interface. In this process, several issues were resolved in both the core KiteFAST code, the controller interface, and the control system package. The main outstanding items include an error in the tether tension, and kite acceleration noise that trickles into the controller infrastructure.

Several comments provided in this document can be summarized as the fundamental need for the inclusion in the development of the aeroelastic code of a subject-matter expert on the specific control system.

The statements in this document are opinions of the author of this report, and do not necessarily represent the opinions of Makani or any other party involved. Whereas they could read as 'harsh criticism', the intent is not to criticize, in hindsight, efforts carried out on a complicated and highly sophisticated system development, but to provide some guidance toward a more efficient development in similar future efforts.

Although external circumstances brought this project to an end prematurely, we are confident that the code is in a state that can make it a viable design tool for future AWE kites, and that with a little more effort on the controller development and aero-structural verification, the kite simulations would be quite accurate in their predictions.

Unknowns still exist on the efficiency of AWE, but RRD is proud of its contribution to the development and engineering of what can arguably be considered AWE's state-of-the-art technology.

RRD Engineering, LLC welcomes any opportunity for collaboration in the future of AWE as well as conventional wind power.

ACKNOWLEDGMENTS

It was an absolute pleasure working with Makani and its staff, and we are thankful for the support received. We thank Dr. Fort Felker for trusting RRD with this critical development task, and for his support and direction. We wish all Makani outstanding staff continued success in their future endeavors. Special thanks go to Ruth H. Marsh, Systems and Product Lead, whose guidance, knowledge, professionalism, support, and unrivaled enthusiasm for clean wind energy made this project not only possible but a success.

We thank Greg Hayman of Greg Hayman Consulting, for providing an overview of the control API work done in the initial KiteFAST development and for being available to answer code organization and compilation questions. We also recognize the support of Rafael Mudafort, Andy Platte, and Jason Jonkman of NREL, for helping with getting the code repository organized, supporting the troubleshooting of different libraries and the gnu compiler on the Linux box, and for suggesting possible strategies for debugging and improving the code.

TABLE OF CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

Ab	Kite MIP absolute acceleration in body(kite)-based CS
Acc_norm	Norm of the kite MIP acceleration
aeroTorq	Aerodynamic Torque at the rotors
Ag	Kite MIP absolute acceleration in ground-based CS
apparent_wind	Apparent wind vector expressed in body(kite) CS or in body-spherical CS
CS	Coordinate system, or reference frame
DCM, dcm	Direction cosine matrix, or transformation matrix between two CSs
Dcm_g2b	DCM from ground to body CS
genTorq	Generator torque
MIP	Most-important-point, reference point in the kite
M600	Makani's Kite
Pqr	Rotational velocities of the kite expressed in body(kite) CS
Rho	Air density
RRD	RRD Engineering, LLC
STI	Systems Technology Inc.
tether_forceb	Tether force imparted onto the kite in body(kite) CS
Vb	Kite MIP absolute velocity in body(kite)-based CS
Vg	Kite MIP absolute velocity in ground-based CS
wind_g	Wind velocity vector in ground-based CS
Xg	Kite MIP coordinates in ground-based CS

1. PROJECT INTRODUCTION AND BACKGROUND

KiteFAST is a new aero-hydro-servo-elastic software code for the dynamic simulation of airborne wind energy (AWE) aircraft (or kites). The core of KiteFAST is MBDyn, an open-source multi-body dynamics code developed by the Polytechnic University of Milan. Other modules that make up KiteFAST were developed by NREL, STI, Makani Technologies LLC (Makani), and RRD Engineering, LLC (RRD). The entire development of this software was solely funded by Makani. The simulation of Makani AWE kite models represents the very first time that KiteFAST was exercised with all its modules enabled and interacting together. The main modules are: InflowWind (to process the wind field and pass it to KiteAerodyn), KiteAerodyn (the aerodynamics force calculator), HydroDyn (the hydrodynamics force calculator), MoorDyn (the module for mooring (tether) line dynamics), and KiteFASTController-Controller (hereafter referred to as *STIctrl*, i.e. the control system set of routines which are specifically designed for Makani's AWE kite). One additional module is *KiteFASTController*, which is an interface to any external control system module that the user would provide. All these modules (except for STIctrl) interact together and with the main dynamics integrator (MBDyn) via a "glue-code", or a set of Fortran and C++ routines that interface the various pieces of software.

RRD was contracted by Makani to support the integration of the controller into KiteFAST. RRD's principal was one of the lead developers of the original KiteFAST code while he was on NREL's staff. The scope of work focused on getting the KiteFASTController module (hereafter KFC) to a more complete and usable interface and on debugging and improving other aspects of the glue-code. The goal of this effort was to provide a functioning software to Makani for the load assessment, development, and certification of the current and next generation kite technology. The development efforts within this project focused on the current M600 kite model.

Whereas other modules (KiteAerodyn in particular) had been verified in stand-alone mode against ASWING and other Makani proprietary data, the entire KiteFAST simulation software had not delivered reasonable data that could be compared to Makani's main kite simulator software (CSIM).

CSIM is composed of a series of python and C++ routines that can simulate the kite dynamics in any phase of flight (e.g., launch and land, tether reel-in and out, acceleration, deceleration, and crosswind). In contrast, KiteFAST was created assuming it would initially be applied only to the crosswind phase of the flight. CSIM, though extremely sophisticated, does not resolve the structural dynamics of the kite (except for the tether and rigid body motions of the kite and ground station) and makes use of pre-calculated aerodynamic derivatives to calculate the kite rigid body dynamics. KiteFAST was envisioned as a tool that would fill this gap and at the same time rely on the CSIM control system structure. Systems Technology Inc. (STI) had been tasked with extracting the needed controller features and software from CSIM and packaging them so that they could be interfaced to KFC.

The complete KiteFAST assembly did not produce successful results, in that the kite typical crosswind flight seen in CSIM could not be reproduced, and the kite would hit the simulated ground plane prematurely.

During the course of this project, several aspects of the KiteFAST code were improved or developed entirely anew, including a completely redesigned KFC with its own driver, improvements and fixes to STIctrl and its interface, bug fixes to the glue-code, expansion of the available output to log important telemetry channels (initially omitted in KiteFAST's development) and improvements to the aerodynamics based on Makani's models. In parallel to these advances, a number of tools were developed to post-process the states and the outputs of CSIM and KiteFAST to support the troubleshooting and verification.

Considerable progress was achieved and a plan for subsequent steps was presented to Makani in February 2020. At that time, Makani encountered challenges that required bringing this project to a close. It was decided that an effort would then be made to summarize the status of the KiteFAST code, lessons learned in the development, and to offer a short user's guide to the new features of the code.

In this document, highlights of the work accomplished by RRD are described in Section 2. Section 3 contains several comments gathered upon the experience in KiteFAST's development and troubleshooting that could be used as guidelines for similar development efforts in the future. A statement on the status of the code to the best of our knowledge and recommended work items for future development are given in Section 4. In the Appendices, a succinct user's guide on KFC's and KFC driver's input/output files is provided.

2. SUMMARY OF WORK CONDUCTED

The work commenced in October 2019 and terminated in April 2020.

The development was carried out primarily on a Linux box, running Debian version 9 (Stretch). The development platform caused quite a few problems, due to the need to manually install and troubleshoot ad-hoc packages and libraries for various tools.

Two Google-Docs documents (Tips&Tricks.docx and ThoughtsOnController.docx) provide both a series of tips and solutions for dealing with these computer and software difficulties and a detailed journal of the progress performed in the course of this project. These documents are accessible by Makani personnel.

The first part of the work consisted in getting the software repository and tools necessary for compiling and running the software (e.g., KiteFAST, CSIM, Blender, and GNU compiler among others). After getting the workstation set up and configured to build and run the various software (which required overcoming several issues with a lot of effort as discussed also in Section 3), CSIM was run to produce benchmark test cases.

2.1. MAIN TEST CASES

Several test cases were generated, but it was soon realized that targeted CSIM simulations had to be created to be used as benchmarks for the verification process. Two CSIM simulations were run with a constant and uniform wind field (10 and 12 m/s winds were tested but no significant difference in the verification results was observed); the states and outputs of the CSIM controller were stored for two complete kite loops. The starting points of the kite cross-wind flights were arbitrarily assumed at around 5 o'clock ('low-start') and 11 o'clock ('hi-start'), respectively. These cases are denoted as *RRD_landbased (and RRD_landbased_free)* and *RRD_land_hiStart (and RRD_land_hiStart_free)*. The two cases were replicated with KiteFAST simulations and run in two modes: 1.) constrained, i.e. with a prescribed trajectory and orientation of the kite following those output by the CSIM simulations; and 2.) unconstrained, i.e. with the kite being controlled solely through STIctrl. The constrained case formed the primary comparison benchmark to improve the KFC API and KiteFAST as whole; however, the accelerations returned by MBDyn with a prescribed trajectory are meaningless. Questions to the Polytechnic University of Milan on this issue have not found an answer at the time of this writing.

2.2. THE CONTROLLER API

An in-depth review of the controller application programming interface (API) (*KiteFASTController.f90* and the C counterpart *KiteFASTController.c*) was carried out. The two pieces of software make up the handshaking and piping between KiteFAST and the subset of CSIM control routines (STIctrl) that was extracted and packaged by STI to function as the actual controller for KiteFAST. These controller interfaces had been developed upon a flawed foundation, i.e. the fact that KiteFAST had been developed without following Makani and CSIM reference frames and conventions. This was mainly due to the limited understanding by the parties involved of the other party's algorithms and goals. As a result, many bugs had been introduced into the codes. In this study, these bugs were identified and (at least for the controller interface) fixed; in parallel to this effort, the convention already established at Makani for flaps and motor/rotor ordering was restored. For example, signs for the flap deflections (that were wrong) and flap index ordering (that was also incorrect thus masking the problem) were corrected in the controller API. Rudder and elevator had also been mistakenly swapped in the piping between the controller and KiteFAST and this was rectified.

2.3. THE (JUPYTER) NOTEBOOK

To analyze and verify the inputs, states, and outputs of the CSIM, KiteFAST, and STIctrl control routines, a jupyter python notebook was created (*KFASTvsCSIM.ipynb*). The notebook is quite extensive and allows for the pre- and post-processing of simulation runs from both KiteFAST and CSIM. The reader is referred to the notebook for comments and explanations of the various cells and variables, whereas here, just a simple overview is provided.

The notebook compares the key controller states (Xg, Vg, Vb, Ag, Ab, acc_norm, dcm_g2b, pqr, tether_forceb_c, wind_g, apparent_wind, rho, aeroTorq, genTorq)[1] as generated by KiteFAST/STIctrl and CSIM. Additionally, the notebook offers an analysis of the controller outputs (commanded flap deflections, rotor speeds, generator torques) and inner control loop commands (e.g., k_geom_cmd, k_aero_cmd, curvature commands).

[1] See List of Symbols and Abbreviations for the meaning of these parameters.

The notebook can also produce input conditions for KiteFAST based on simulations performed in CSIM. The initial conditions include the MIP (the main reference point of the kite, taken coincident with the kite origin, and to which the main kite reference frame is fixed) position, orientation (in rotational vectors, see MBDyn's User's Guides for definitions), translational and rotational velocities in the inertial reference frame, and the initial control settings (flap deflections, generator torques, rotor speeds, and rotor blade pitches at t=0s). These can then be input into the MBDyn input files (.mbd and .set files), MoorDyn's and KFC's input files (see also Appendix A).

Finally, the notebook can also be used to generate data files of controller states as well as controller outputs. These files can be read in by KFC and its stand-alone driver as discussed in Sections 2.4-- 2.5.

2.4. MISCELLANEOUS CODE IMPROVEMENTS

Through the use of the notebook, many of the bugs and flaws of KFC, the controller API, and KiteFAST were discovered. However, several improvements had to be performed in parallel to bug fixes to obtain the necessary output channels from KiteFAST and KFC. KiteFAST (namely within the glue-code, KiteAeroDyn, and KFC) was augmented to include output channels such as:

- From the glue-code: KiteVxi, KiteVyi, KiteVzi (MIP velocity components in ground-based reference frame), KiteTAxi, KiteTAyi, KiteTAzi (MIP acceleration components in ground-based reference frame), KiteRVxi, KiteRVyi, KiteRVzi (MIP rotational velocity components in ground-based reference frame);
- From KiteAerodyn: AirfFxi, AirfFyi, AirfFzi (aerodynamic forces and moments on the airframe (no rotor actions included) in the ground-based reference frame)
- From KFC: Rot1-8GTq (generator torques), Rot1-8Spd (rotor speeds), Flp1-8Def (flap, rudder and elevator deflections), DCMG2Bc1-9 (dcm_g2b_c); and TethFxb, TethFyb, TethFzb (tether_force_b in kite CS). See also Section 2.4 and Appendix C for more information. Note that flap deflections could also be obtained through KiteAerodyn, but these are direct pass throughs from the STIctrl and more relevant to have in one common place, i.e. the KFC API.

All these new output channels are absolutely necessary in the development of any kite technology and allow for a more complete analysis of the kite dynamics and comparison to CSIM.

2.1. UPDATE OF TETHER FORCE

The tether force variable passed to the STIctrl (tether_force_b), which by CSIM design should be the bridle knot tension vector, is not directly available from the MoorDyn module. As such, it was originally approximated by the vector sum of the tension forces at the fairleads (connections to the aircraft). This is an aspect that could be improved in the future, by creating the needed output in MoorDyn. The approximated value, however, can now be analyzed as a channel in the KFC output files.

It was also noted that the value passed to the KFC API and STIctrl lagged by 1 controller time step. This was changed in the glue-code but should be revised in the future to assess the order of the submodule calling within KiteFAST. This fix had an effect that was most visible in the simulation initial time step. At the end of the project, attention was paid to MoorDyn's fluid density, and it was noted that a bug existed where the KiteAerodyn density was not being passed properly to MoorDyn. Fixing (Rafael Mudafort of NREL carried out this fix) this value significantly reduced the main tether ~1Hz vibration amplitude. However, the tether force that is being used in STIctrl derives from the bridle lines, which still show considerable noise in their tension signals.

2.2. ROTOR/MOTOR IMPROVEMENTS AND FIXES

Beside those mentioned elsewhere, another major issue was found in the motor/generator model within STIctrl. This model was created by Makani's staff to try and replace the rotor control model in CSIM. It was decided at the beginning of KiteFAST's development, that MBDyn would not be burdened with the rotor dynamics integration. The main issue was thought to be the very high rotor rotation speed (very high frequency) and the consequent need for very small time-steps to ensure the code would converge at each time step. The STIctrl generator model was not returning the generator torque to the KFC API but the sum of both the aerodynamic torque (calculated by KiteAerodyn) and the generator torque (locally calculated). This error derived from a misunderstanding of the KFC API interface and KiteFAST organization and requirements. With this bug fixed, the rotor related quantities (speeds, torques, thrust) matched much better between KiteFAST and CSIM for the simulated test cases. It is important to note that the model is a crude simplification of a much more complicated motor control module within CSIM, and that the integration used to calculate rotor accelerations and speeds is likely too coarse and should be improved in the future.

2.3. KITEAERODYN IMPROVEMENTS AND FIXES

To further improve the rotor related dynamics, the air velocity at the rotor hubs was modified following a model recommended by Makani that accounts for the circulation about the wing surface. Note that this is hardcoded within KiteAerodyn's actuator disk module. In the future, this should be improved to allow for the user input of the model parameter settings.

An important oversight (or misunderstanding) was found in KiteAerodyn's actuator-disk model. The bug was the result of the lack of enforcement of clear prescriptions and conventions at the time of the definition of the 'design basis' for the new software. The skew-angle was calculated as the 180-deg complementary angle of what CSIM assumes, therefore thrust values had wrong signs. This was fixed, and the thrust values now match relatively closely those from CSIM in the constrained test cases.

The new output channels (AirFx-zi, AirMx-zi) can be used to assess the aerodynamic model in KiteFAST vs. CSIM's. The results showed that, after all these fixes, the z-component of the airframe force is very well captured by KiteFAST, but differences existed in the y and x components of the airframe loading.

Makani recommended increasing the model aerodynamic drag with an additional 0.075 factor to sum to all calculated Cd's. This was hardcoded in KiteAerodyn; in the future, it should be left as a user-input correction factor for the drag calculation. With this modification, the airframe force along the kite x-axis matched much better that from CSIM, though KiteFAST still overestimates this propulsive force (see Figure 1).

The y-component of the airframe loading was also improved by modifying the airfoil description of the pylons, yet the sign of KiteFAST's prediction is suspicious.

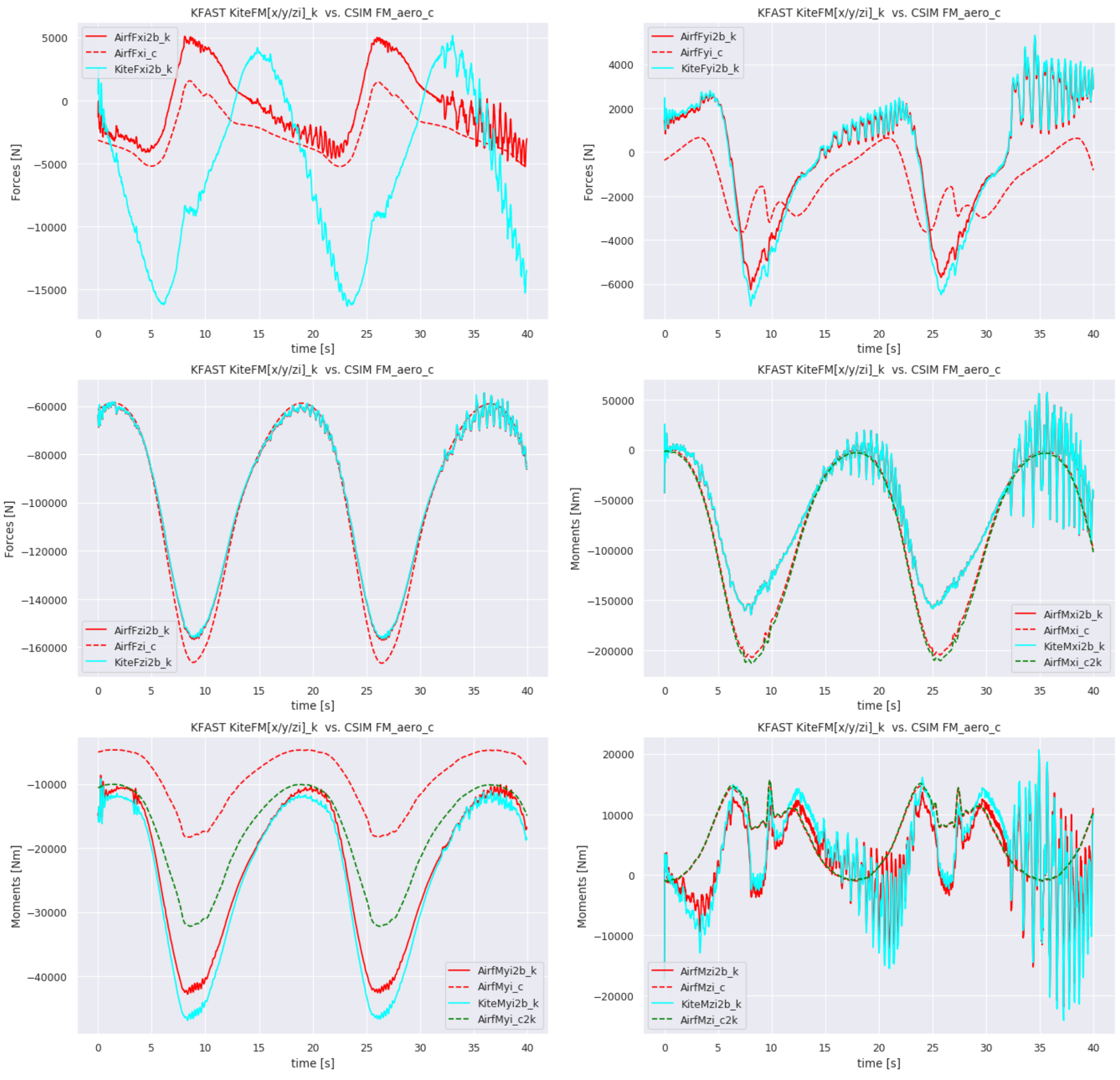


Figure 1. Airframe aerodynamic forces and moments along kite coordinate system x, y, and z axes; legend: solid red lines are KiteFAST results; dashed lines are CSIM results; the cyan lines show airframe forces and moments that include rotor aerodynamic actions. These results were obtained for the RRD_land_hiStart case and with commanded control settings from CSIM outputs.

2.4. KFC REDESIGN

KFC (KiteFASTController.f90) was completely redesigned. In addition, a flag (KFCmode) was introduced in KiteFAST and the MBDyn user module element to allow for three different ways of exercising the controller API. KFCmode (which must be specified in the .mbd file user-module section) can be set equal to 0, in which case the simulation will not use the STIctrl, but rather assume constant (hardcoded) control outputs (see KiteFASTController.f90). With KFCmode=1, the normal API to STIctrl is enabled, and an input file for KFC must also be specified. With KFCmode=2, the API enables the simulation to bypass STIctrl and to read in values for the control outputs from a data file specified in the KFC input file.

KFC is now capable of reading in an input file. This allows for certain parameters to be passed at initialization, including the file location of the controller dynamic library, time step, and initial conditions for the controller outputs. These were hardcoded in a 'global' structure within the STIctrl that also contained many other parameters, and variables that would store filtered quantities of the states transferred by the KFC API. The plan was to completely remove this global structure. Currently, only the mentioned major variables have been replaced by the correct ones (which are directly transferred from KiteFAST). The KFC input file also allows the user to specify a data file ('Dummy-Controller Data File') to be used with KFCmode=2 as stated above. By using this mode, one can assess certain dynamic characteristics of the kite under specified control commands, while removing the 'volatility' associated with a 'black-box' controller. Furthermore, this mode can be used to simulate special cases, such as faults in the ailerons or other control chain actuation, and special maneuver cases that may be required for certification.

KFC can now also write to a file its time series of output channels.For more details see Appendix A.

2.5. KFC DRIVER

KFC can also be run in stand-alone mode. A KFC driver routine was created to run the STIctrl separately from KiteFAST. In this fashion, the STIctrl can be verified against CSIM, for example, after feeding it the exact same states as those used by CSIM. The notebook can be used to generate a data file containing time series of all the states that normally would be computed by KiteFAST and passed via KFC to STIctrl. The typical KFC driver input file can be seen in Appendix C and the typical states data file can be seen in Appendix D.

2.1. THE STIVSCSIM.IPYNB NOTEBOOK

Running the KFC API in stand-alone fashion allowed for the vetting of the STIctrl. A new notebook (*STIvsCSIM.ipynb*) was created to assess the STIctrl states and outputs vs. CSIM states and outputs. The reader is referred to that notebook for a comprehensive review of the results. Herewith, we summarize the observations. Note that a few bugs were identified and fixed within STICtrl during this new simulation run, though they are not deemed critical to the simulation outcome.

The states used by CSIM in the typical constrained test case were fed to the KFC API and the notebook showed that they matched exactly in most cases, except for some filtered quantities in the STIctrl that showed some slight lag when compared to the analogous quantities from CSIM. These differences are deemed negligible for this level of comparison. The outputs, however, do not match very well. The generator torques and the rotor speeds are generally better behaved than the flap deflections (see Figure 2), but overall, this confirms that even with the 'best possible' states passed to STIctrl, the outputs still do not match those from CSIM. Future work should focus on overhauling STIctrl.

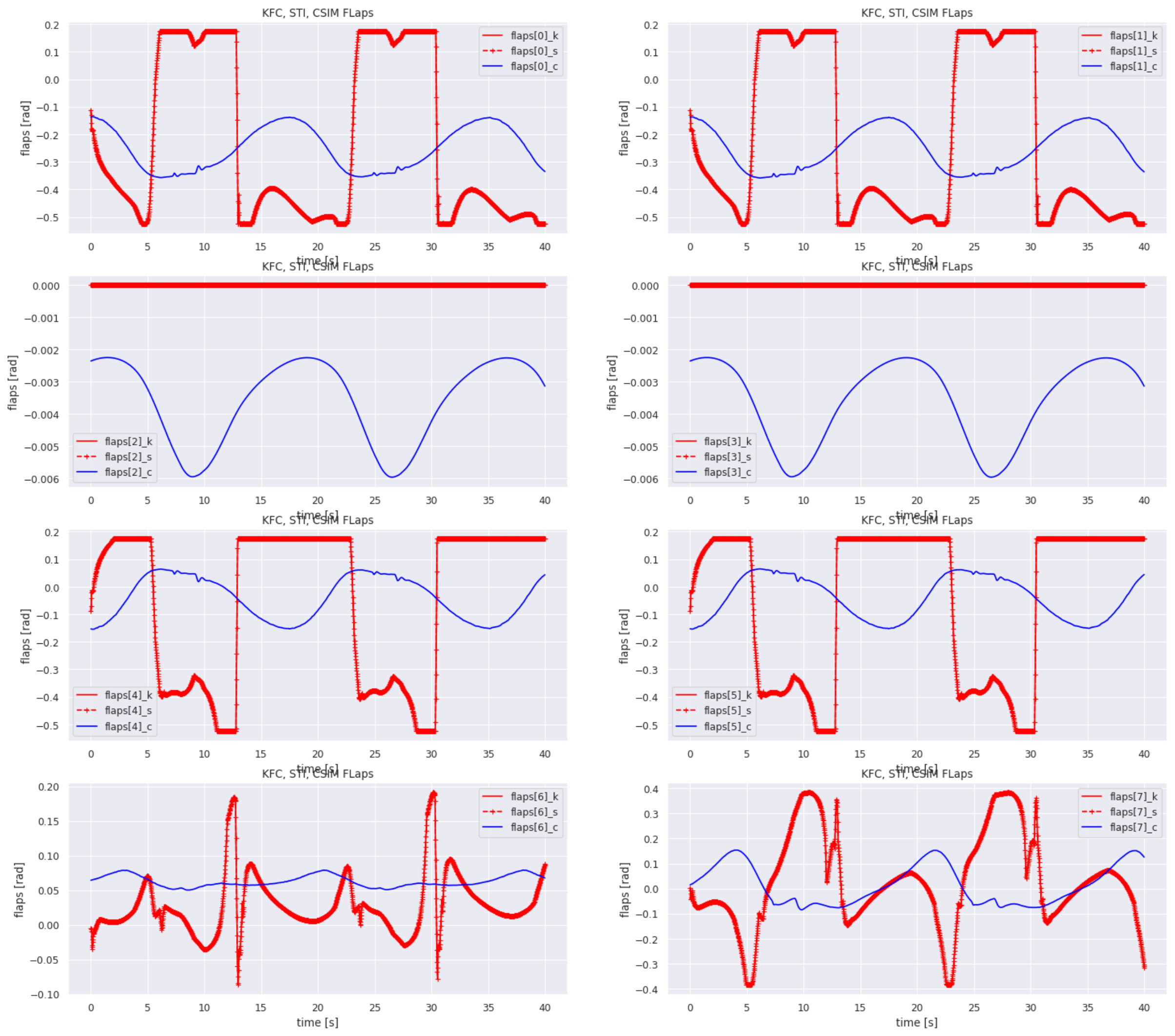


Figure 2. Flap deflections; legend: red lines are KiteFAST results; blue lines are CSIM results. These results were obtained for a standalone KFC driver simulation, where the states were obtained from the same CSIM run as that in the RRD_land_hiStart case. Note that the initial conditions were not properly matched in this case, but this should not affect overall results.

3. LESSONS LEARNED

Discussing all the lessons learned during this project would require hours of interviewing of most of the individuals involved. Here, we provide the key factors that have been identified by RRD and that could make the next phase of development or a similar effort more efficient. These are the opinions of the author of this report, and do not necessarily represent the opinions of Makani or any other party involved. It should be noted that these lessons were learned in the context of a software development that was outsourced (by Makani to NREL) and that did not have strict specifications in terms of interfacing to the existing (Makani's) simulation convention sets and infrastructure. As a result, the software was built upon a different infrastructure, i.e. one made for wind turbines, and not for AWE kites. Adapting one to the other caused several unintended consequences. Additionally, the comments in this Section are generated with the assumption that the aeroelastic code KiteFAST was commissioned to be created for the development of a specific technology and to serve the commissioning company (Makani).

In the following subsections, a brief summary of each topic is provided together with recommendations that could be followed if a similar development effort were to take place again within the same circumstances.

3.1. USE OF AN 'UNSUPPORTED' OS PLATFORM

Though Linux is an accepted platform for code development, the Debian Stretch distribution demonstrated to be possibly the worst platform for multi-language code development and verification efforts via integration of widely used computational and visualization packages. The lack of up-to-date support (as opposed to the Ubuntu distribution for example) makes integrating the latest software packages and compilers extremely difficult and time consuming. Individual libraries need be downloaded, built, and installed manually. For example, several workarounds needed be devised to install the latest Blender and Blendyn visualization software or the more recent versions of the GNU compilers. This set-up is more viable for computer science than active engineering development. Additionally, the platform we used for this study did not allow us to work in double precision, likely due to some corrupted library that could not be recovered.

Recommendation: Use a widely supported distribution of the operating system.

3.2. DIFFERENT REFERENCE FRAMES

KiteFAST uses a ground coordinate system where z points upward, and x downwind. CSIM and STICtrl adopted a reference frame where z points downward and x upwind. This is possibly the single largest contribution to confusion and unnecessary mathematical strain in the development of the interface between the main code and the controller. The ramifications of this choice created more than one problem in almost every aspect of the troubleshooting and verification, from the definition of initial conditions to the analysis of the Euler angles. As can be deduced from comments left in the various pieces of codes, different consultants interpreted rotations and Euler angles differently, and in some cases mistakenly; significant effort was spent on trying and reconciling the orientation matrices from KiteFAST to CSIM especially in setting up the constrained test cases. Some lingering problems with the tether force (x component showing a suspicious positive value) and possibly other channels, may still be related to the choice of the main coordinate system which requires the Euler pitch angle to be 180° offset from the CSIM one. This aspect remains to be fully troubleshooted.

Recommendation: One single and common reference frame (or set of coordinate systems) between the controller and the aeroelastic code should be used to avoid unintended consequences and tedious bugs.

3.3. ADAPT CONTROLLER TO KITEFAST OR ADAPT KITEFAST TO THE CONTROLLER REQUIREMENTS?

Beside reference frames, several KiteFAST conventions were chosen in conflict with the CSIM ones. Whereas this could be acceptable if KiteFAST were to be developed in a 'vacuum' and without the underlying requirement of interfacing to Makani's CSIM (specific controller or kite model), in the opposite case (which was indeed that of the initial KiteFAST development) this was destined to cause a number of issues and mistakes. Several bugs introduced in the initial development of KiteFAST had to be fixed in the course of this study. To avoid these issues, an aeroelastic code designed to support an existing AWE project should replicate all the reference frames and index ordering of the main components (e.g., rotors/motors/pylons, command surfaces) as per the established controller and kite model (Makani's CSIM conventions in this case).

Recommendation: To the maximum extent possible, the aeroelastic code should adapt to AWE's prescriptions and not vice versa.

3.4. COMPLEXITY OF THE BASIC CSIM CONTROLLER

CSIM is a highly sophisticated software program, and it is made up of two fundamental components: the 'estimator' and the 'controller'. CSIM was devised for the accurate simulation of the kite dynamics in all phases of flight. It contains several hardcoded parameter values, and configurations settings that are very specific to the M600 kite (Makani's main kite model) and its testing site environments. The two components are intimately connected to each other and separating one from the other is arduous and most likely prone to generating unintended results.

Makani staff explained that the estimator has two primary functions. First, its "core" turns raw sensor data (GPS positions, IMU data, loadcell readings, pressures seen at the six ports on the Pitot probe, etc.) into the more abstract "state estimates" in engineering units: e.g., position (Xg), velocity (Vg), acceleration (Ag), attitude (dcm_g2b), body rates (pqr), tether tension vector, and apparent wind vector, all in real units. Second, the estimator also computes many derived values, such as low-pass-filtered versions of the raw estimate from step 1, and more complex combinations such as the output of the complementary filter for apparent wind.

For the initial specification of the M600 controller for KiteFAST, Makani directed STI to implement it as simply as possible. The STIctrl includes only the controller itself, and bypasses both the estimator core and second estimator step. This turned out to be an oversimplification, however, and did not work. STIctrl did try to compensate for 'short-circuiting' the second estimator step by filtering some variables, albeit with incorrect settings. At that time it was not fully understood that, "The filters of the estimator step 2 are unambiguously part of the control laws" (Tobin Fricke, Controls Engineer, Makani), and therefore must be implemented if the M600 controller is to function properly in KiteFAST.

While initially the thought was that it would *not* be necessary to model the sensors themselves (as is done in CSIM), since KiteFAST can provide all the precise physical states (i.e., the "ground truth") needed by the controller, "It seems like it could be very hard to decouple parts of the estimator that are 'sensor to ground truth' vs 'ground truth to filtering/processed data'" (Michael Scarito, Controls Engineer, Makani).

Overall, the controller was found to be extremely 'tight', i.e. tuned to the idiosyncrasies of the Makani M600. Inevitably, a different discretization and parametrization of the physics of the kite would generate conflicts with the controller. The CSIM controller has multiple nested inner loop functions that create very tight tolerances on the commanded outputs. Because of this complexity, and because the project terminated early, there was not sufficient time to delve into the controller to understand which variables could be most critical and sensitive for the outputs.

Recommendation: Start with a generic and simpler controller that could work with a generic kite. Avoid hardcoded parameters within the controller and leave everything to input files or in-memory variables to be passed via the aeroelastic code. Spend more time clarifying the controller main functions and identifying the critical variables that dictate the command laws.

3.5. DEVELOPMENT OF THE AEROELASTIC CODE WITHOUT THE FULL UNDERSTANDING OF CSIM'S CONTROLLER ALGORITHM

KiteFAST and its interface to the controller were originally developed without the necessary knowledge of the inner workings of CSIM. This generated all the subsequent problems that are also listed here.

"My feeling is that we need a Controls/Software Engineer with a tight rein on STI/NREL if we want to get usable results" (Tobin Fricke, Makani). This did not happen primarily as a result of constraints on resourcing – until near the end of the KiteFAST project the Makani Controls team priorities were entirely (and justifiably) focused on control system upgrades in support of the on-shore and off-shore M600 flight test campaigns.

Recommendation: Develop the aeroelastic code in direct contact with the controls engineering team, better yet have a person dedicated to the interface between the aeroelastic code development team and the controls team to develop enough knowledge of both and answer questions by both parties and enforce requirements. The controls team must be an integral part of the development, the requirements on the aeroelastic codes should be tightly controlled by the AWE infrastructure already in place and not vice versa.

3.6. SENSOR LOCATION

Though some consideration may have gone into the location of the IMU and GPS during the development of KiteFAST, there are subtleties associated with how the controller processes data. CSIM assumes a certain position of the pitot tube (together with a bias for its orientation on the airframe), GPS, and IMU, and corrects for rigid body motion at those locations to obtain a picture of the relative air velocity vector (analogously for the accelerations and rotational velocities). These subtleties create slight differences in the calculated states between KiteFAST and CSIM, and especially in the angle of attack and sideslip angles.

Makani engineers explained clearly that the ‘apparent wind vector’ must be calculated at the exact air data probe location and injected into the estimator before the estimator re-references that vector to body coordinate system and accounts for body rates. Similarly, the accelerations and rotational velocities must be calculated at the IMUs and the GPS position, respectively (analogously for body velocity). Then the estimator should combine these values. “The Kalman filter needs to be "in the loop," not so much because of transfer function considerations, but because it incorporates data from sensors at different locations on the kite” (Tobin Fricke, Makani).

Recommendation: Ensure the sensor location is given the required importance in the development of the aeroelastic code or correct for the lack of that information in the controller and create a more versatile controller that can get information tied to other nodes in the aeroelastic model.

3.7. THE IMPORTANCE OF THE TETHER

The tether exerts the largest forces on the kite and the controller relies on tether tension and orientation to formulate laws for the commanded output. As such, the tether dynamics is an integral part of Makani’s controller. As CSIM assumes the tether is always attached to the kite, except in fault cases, there is no possibility of simulating straight and level flights. This poses a fundamental problem in the development and verification of the aeroelastic code. Debugging and visualizing dynamics in crosswind are at least an order of magnitude more difficult than in straight and level flight.

Additionally, the importance that the tether plays was likely not fully understood at the beginning of KiteFAST’s development, and MoorDyn was not expanded enough to directly provide the needed bridle knot force to the controller. In its place, an approximation was made whereby the fairlead tensions from the bridles were summed vectorially and then transformed into the kite CS. CSIM feeds the derived quantity ‘tether_roll’ to its inner control loops, thus KiteFAST’s calculated vector had to be further manipulated in STICtrl to first transform it into the kite spherical CS, then filter the roll angle, and finally pass it to the inner workings of the controller. Some problems have been found with this series of transformation and were fixed, but it appears as the x-component of the tether tension coming from KiteFAST still returns the wrong sign.

Recommendation: Create a controller that can operate in straight and level flight with and without tether, and with and without active motors. This will allow for much easier verification efforts and will help debug and produce better aerodynamics and structural dynamics models. Make provisions for the controls to operate with untethered flights, even though that means simulating an impossible situation, because it makes the software and model verification/validation process more effective.

3.8. ACCELERATIONS AND POSSIBILITIES OF MISTAKES

STIctrl required a series of accelerations to be passed from KiteFAST to STIctrl. These are virtually the same accelerations (i.e. the MIP acceleration with respect to the inertial reference frame) but expressed in different coordinate systems and also provided as vector magnitude. Simultaneously, the kite orientation matrix is also provided to the controller. This information is superfluous, and it makes little sense that all of these are piped downstream. It creates unnecessary redundancy, memory waste (though minimal), and especially the possibility for confusion and mistakes. Many times, the question arose as to which of these quantities are filtered by the estimator, and which of these quantities really drive the controller's inner laws. This was never fundamentally put to rest.

Recommendation: Reduce the number of states to the bare minimum needed for the controller to run its internal laws, reduce superfluous bookkeeping, and reduce the possibilities for misunderstandings and bugs.

3.9. MOTOR/GENERATOR MODEL SIMPLIFICATION

The motor control within CSIM was fundamentally different from how a wind turbine's torque control is usually modeled. This created confusion at the beginning of KiteFAST's development and numerous iterations were required to come to an agreement on how to best handle the motor/generator dynamics. Because of assumed limitations in MBDyn (namely associated with the need for very small time-steps to simulate very high rotor RPM), the burden to interface aerodynamic torque to generator torque and to solve for the rotor speed was left to the STIctrl. This is a reasonable approach, but the first order integrator used in the STIctrl might be too crude of an approximation and create some incompatibility with the integration scheme used throughout the KiteFAST code.

Recommendation: Assess whether the MBDyn time step requirement could be bypassed and the rotor dynamics be integrated within MBDyn itself. Revisit the simplified integrator and assess whether a higher order scheme should instead be adopted to be compatible with the time steps used and the calling order of the submodule chain.

3.10. TRIM CONDITIONS AND NEED FOR STRAIGHT & LEVEL FLIGHT CAPABILITIES

Gliding simulations (untethered) were run toward the end of the project, after Makani suggested a way to simulate a faulty condition in CSIM. The KiteFAST results showed how the kite performs a much tighter spiral when compared to CSIM's. This points to different trim conditions between the two realizations. In KiteFAST, the mass and inertias of the rotors are not directly handled by MBDyn for the integration limitation discussed above. Their inertial and gravitational effects are added via one of the submodules. This makes it more difficult to assess (indirectly) the location of the CG with respect to the aerodynamic center of the aircraft.

Recommendation: Spend sufficient time in the trimming of the kite model within the aeroelastic code, to match the CSIM trim. This should be done exercising the entire aeroelastic code and not just within individual modules run in stand-alone fashion.

3.11. USE OF AN EXTREMELY COMPLEX AEROSTRUCTURAL KITE

KiteFAST was initially verified for the aerodynamics (lift primarily) of isolated kite components (wings, tails, etc.). Then, via KiteAerodyn stand-alone simulations, a simplified model of the kite was also analyzed. The following step should have been a full KiteFAST verification against CSIM with the simplified kite model, progressively moving toward the complete M600 model. The code was never fully vetted with a simpler aircraft (kite) model (simpler aerodynamics and structurally simpler) against Makani's simulator. In addition, a simpler control algorithm should have been used. Given the nature of the established CSIM simulator tool, which is tightly tuned to the specific M600 model, these intermediate steps could not be performed in a short time and would have required a fundamental reformulation of the tool architecture. This should be at all costs avoided in a future endeavor that looks at making use of KiteFAST for design, given that KiteFAST has not been validated against other simpler models at this point.

Recommendation: Plan for a stepped verification phase, where all the aeroelastic features are vetted individually and in concert with the remainder of the code. This may require modifications to the existing simulation tool (e.g., CSIM) or the aeroelastic code, or both. Focus on simpler rigid body dynamics at first and assess trim conditions and response to disturbances and commands. Use incrementally more complicated kite aerodynamic and structural models, making sure each step is successful before increasing complexity.

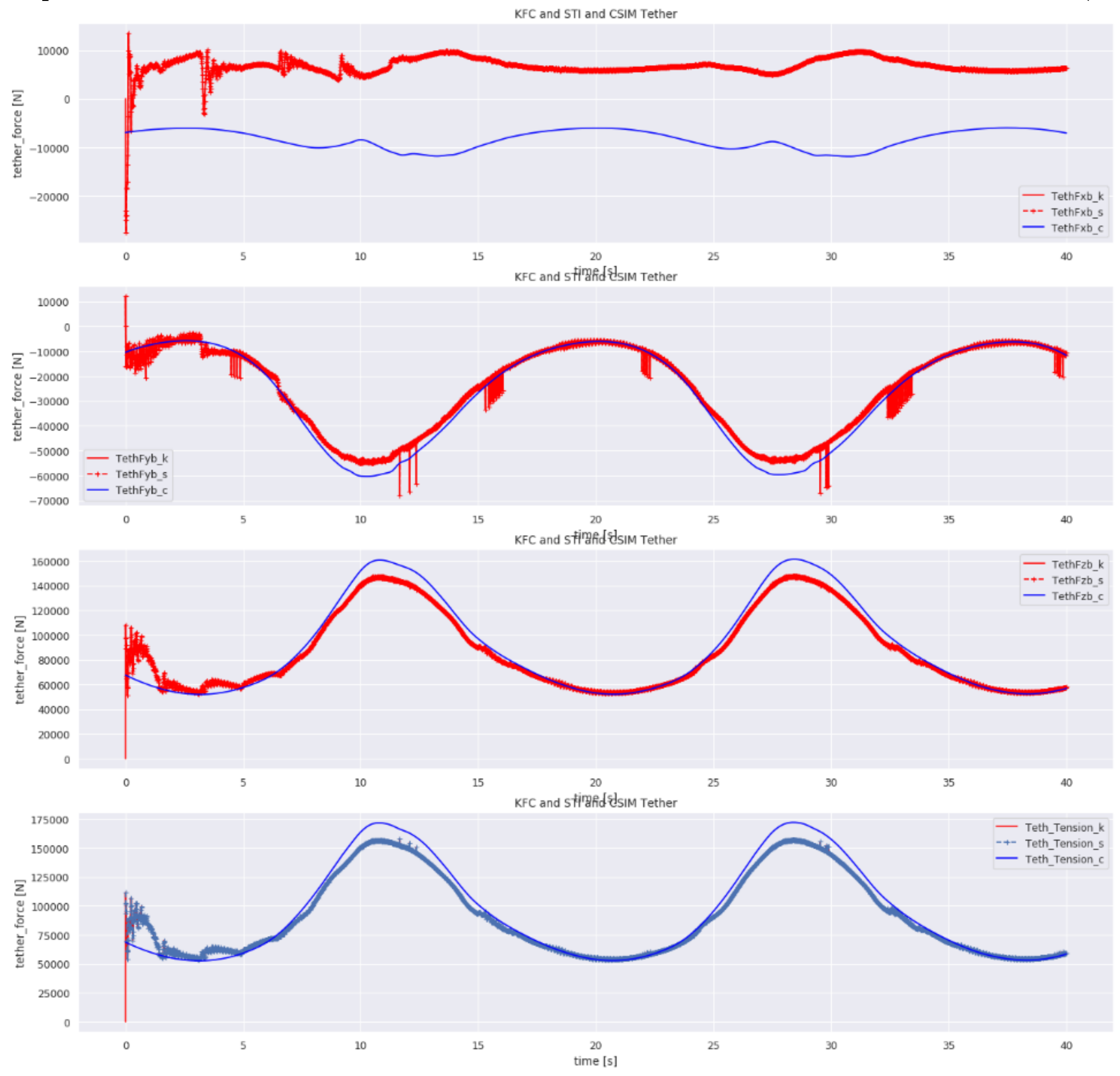
4. STATUS OF KITEFAST AND RECOMMENDED FUTURE PATH

KiteFAST has undergone a rigorous review with emphasis on the KFC API and several bugs were fixed. Auxiliary post-processing software was created, including jupyter notebooks, python scripts, and visualization scripts and tools. New capabilities were introduced to test the KFC API and the STIctrl in stand-alone mode. Special test cases (constrained and unconstrained) were produced in both CSIM and KiteFAST to verify and validate the code.

Makani engineering and empirical knowledge went into some improvements for the aerodynamics, in particular to better capture interference drag and to better describe the relative air velocity at the rotors. Some of these changes are hardcoded into KiteAerodyn, as there was insufficient time to expand user inputs and documentation to make them more versatile.

As a result of the verification efforts vs. CSIM simulations, KiteFAST has reached a level where the most critical bugs are thought to have been removed from the KiteFAST to controller interaction. Below is a list of outstanding items (in the perceived priority order) that should still be addressed and could be seen as a checklist for the near-future development of the code.

The tether force (tether_force_b) component along the x axis of the kite shows the wrong sign when compared to CSIM (see



1. Figure 3). It is expected that a bug may exist either in the MoorDyn module or glue-code. There was not enough time to further investigate this aspect, but the fact that the other two components match CSIM counterparts, points to a potential problem with the transformation from inertial reference frame to kite CS. A lingering 180° complementary offset in the kite 'pitch' angle could lead to this sign change, but it could not be proven.

2. Another aspect that needs some attention is the high frequency noise in the bridle tension signals. The bridles are much stiffer (and shorter) than the main tether. This may indicate a flawed set-up within MoorDyn, perhaps out of the range of applicability of that module.

3. STIctrl, the controller routines used in lieu of the CSIM controller, still contain filters and global structure data that should revisited and corrected. The KFC driver and a pre-set time series of

controller states can be used to exercise the STIctrl in standalone mode and apply needed corrections.

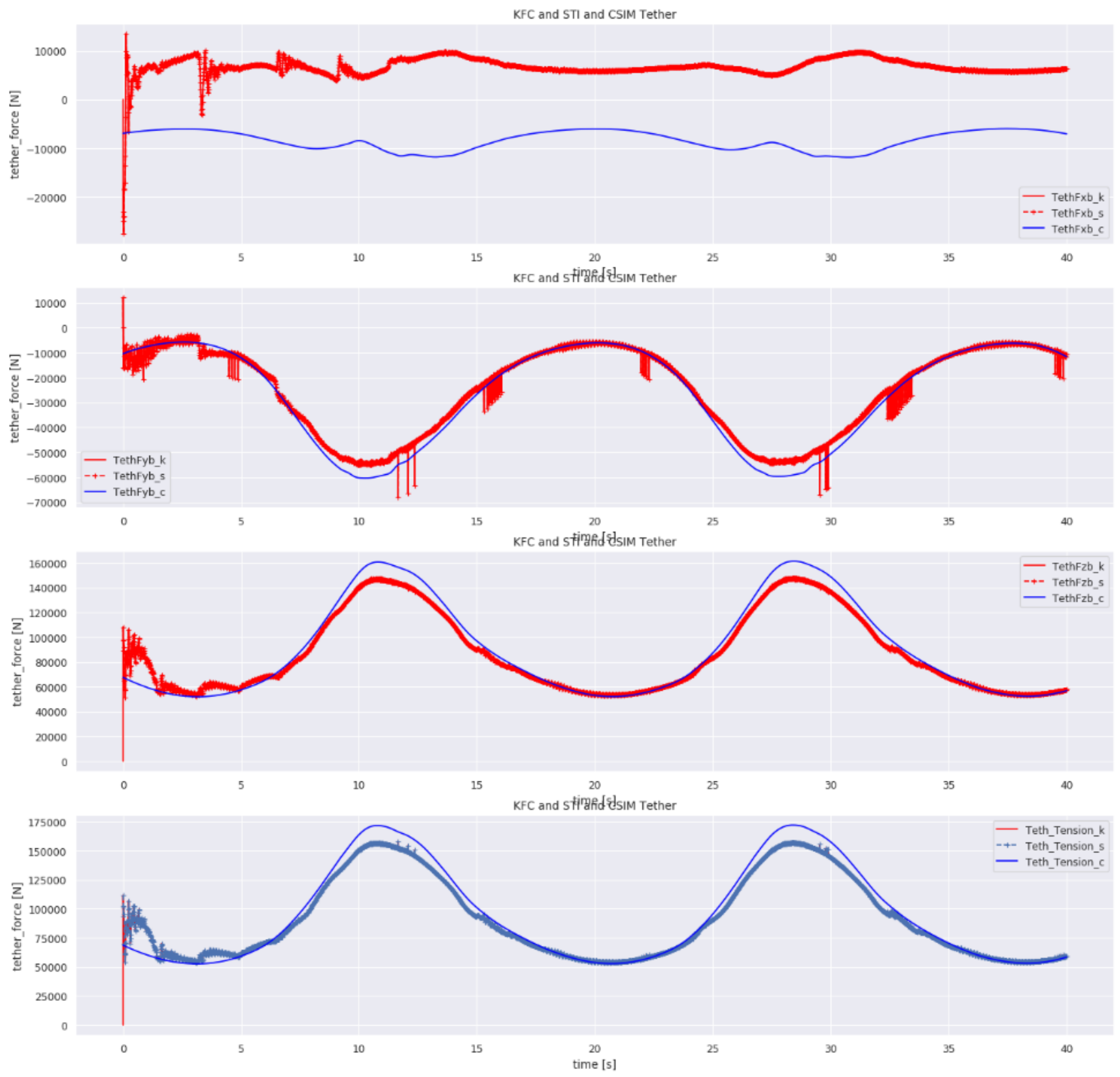


Figure 3. Tether_force_b components in kite coordinate system x, y, and z axes and magnitude; legend: red lines are KiteFAST results; the blue lines are CSIM results. These results were obtained for the RRD_land_hiStart case and with commanded control settings from CSIM outputs

4. The rotor/motor model should be reviewed especially in terms of integration order and compatibility with the underlying integration scheme within KiteFAST.
5. MIP accelerations are quite noisy in the unconstrained case, but nil in the constrained case. This should be revisited with the help of the staff at the Polytechnic of Milan. Verbal communication indicated that the noise may be due to the large tolerance value in the 'problem solution' setting of MBDyn. That may also be the cause for the needed small time-step.

6. The aerodynamic model was tuned to Makani's M600 model. The airframe forces were calculated to be fairly close to those predicted by CSIM, especially in the longitudinal plane dynamics. The match in the lateral force and moment dynamics is, however, poor. This is in part due to the difference in CG location and trimming, and in part due to potential differences in the airfoil discretization for the pylons.

7. The rotor/nacelle mass and inertial modeling should be revisited to ensure that the effect of the masses of the rotors enter in the MBDyn overall integration model. In parallel, one should verify the CG location and trim conditions of the test kite model.

8. The VSM method has problems converging in single precision and to a certain extent caused other numerical convergence issues in double precision. This should be revisited and corrected as needed.
9. The preprocessor should be reviewed and modified as needed to account for the changes performed, to make sure the new capabilities can be captured when setting up a new model (e.g., KFCmode values).

In addition to these items, below is a short list of improvements that would make KiteFAST more robust and powerful:

1. Improve beam element formulation in MBDyn, to include consistent mass treatment. Currently distributed mass must be lumped at beam nodes, and no 6x6 mass matrices can be assigned, which makes the preprocessing quite involved (the author developed the procedure for redistributing the mass from 6x6 cross-sectional matrices to lumped masses while minimizing errors in the inertial properties of the beam segments).
2. Improve aerodynamics possibly with a lattice vortex method. Tracking vortex filaments or particles (depending on the flavor of the vortex method selected) can also account for any rotational and curling effect in the wake, and will increase the fidelity though with some increase in computational demand.
3. Develop a cable element for MBDyn together winching capability. This will improve the fidelity of the tether and bridle dynamics simulation and allow for the simulation of other phases of flight such as launch, reel-in and reel-out.

5. REFERENCES

RRD Engineering, (2019-2020): "ThoughtsOnController.docx", found at **https://drive.google.com/open?id=18jyJVR7PuKrbgEDNePwicFLTM0BK7clc**

RRD Engineering, (2019-2020): "Tips & Tricks RRD.docx", found at https://drive.google.com/open?id=1KEP2AOM-lJUayQidsHos-j7iW1Mget2aZEu8cgQ2izQ

RRD Engineering (2019-2020): "KFASTvsCsim.ipynb", a jupyter notebook found at [KiteFAST repository]/glue-codes/kitefast/test_cases/scripts

RRD Engineering (2019-2020): "STIvsCSIM.ipynb", a jupyter notebook found at [KiteFAST repository]/glue-codes/kitefast/test_cases/scripts

.

APPENDIX A: KITEFASTCONTROLLER.F90 INPUT FILE

The Input File for the entirely redesigned KFC module is shown below.

Each section of the file is prefixed by a heading line. The various inputs are self-explanatory, and herewith we offer a description of only the key parameter settings.

DTctrl is the time interval for the controller calculations. Note that this value must be 0.01s (100Hz) for the standard Makani CSIM controller.

DLL_Filename is the complete path and file name of the controller library. The DEFAULT keyword can be used to point to the default library, which is the STICtrl, compiled and installed in the indicated default path location.

In the 'Initial Conditions' section, the initial values of blade pitches, generator torques, rotor speeds, rotor accelerations, and flap deflections can be declared.

In case the user required KiteFAST to be run with an a-priori calculated set of control commands, a "dummy controller" data file can be specified as DummyCFil. The format of the DummyCFil is shown in Appendix B.

The final section of the KFC input file is dedicated to the required output format and channels. This follows the remainder of KiteFAST convention for output specification and it is not described in detail. The output channels that are now available from KFC are all the control outputs, such as rotor speeds, generator torques, and flap deflections, all in the Makani CSIM ordering and sign conventions. Additionally, the user may request the output of the additional STICtrl states dcm_g2bc and tether_force_b, again in the CSIM convention and reference frames. Note that the 9 DCM components are unraveled in the Fortran column-major order.

```
--- KiteFastController INPUT FILE ---
Testing Controller
--- SIMULATION CONTROL ---
False     Echo      ! Echo input data to <RootName>.ech? (flag)
0.01      DTctrl    ! Time interval for controller calculations (Default=0.01)(s)
DEFAULT   DLL_Filename  ! DLL_Filename   default is "~/sandbox/build/modules/kitefast-controller/libkitefastcontroller_controller.so"
--- INITIAL CONDITIONS ---
0.        0.       0.       0.       0.       0.       0.       0.        rtrBladePitch            !< The rotor blade pitches [8 values] [rad], for all rotors in CSIM order convention
-30.173    21.697    12.640    -3.371    61.978    -68.994    -76.138    83.087  genTorq    !< The generator torques [8 values] [Nm], for all rotors in CSIM order convention.
-147.040   144.957   142.847  -140.691   164.788  -166.670  -168.517   170.323 rtrSpd  !< The rotor speeds [8 values] (rad/s), for all rotors in CSIM order convention.
7.521      -8.332    -9.234    10.085    -9.473     8.807     8.287    -7.706  rtrAcc         !< The rotor accelerations [8 values] (rad/s^2), for all rotors in CSIM order convention.
-0.137     -0.138    -0.002    -0.002    -0.153    -0.152     0.064     0.015    ctrlSettings   !< The control surfaces angles [8 values] [rad], for all surfaces in CSIM order convention.
--- DUMMY CONTROLLER DATA FILE ---
"./RRD_ctrl.outs"     DummyCFil       !  If KFCmode=2 AND this is not blank, it points to a data file that has time series of CTRL output data, and above Initial Conditions will be trumped
--- OUTPUT ---
True                    SumPrint        !  Print summary data to <RootName>.sum? (flag)
1                     OutSwtch          !  Output requested channels to? (-) (switch) {1=KiteFAST.KFC.out, 2=GlueCode.out, 3=three files]
"ES10.3E2"                OutFmt        !  Format used for text tabular output, excluding the time channel; resulting field should be 10 characters (string)
8                     NRotOuts          !  Number of Rotors whose gentroque will be output [0 to 8]
1 2 3  4 5 6 7 8           RotOuts      !  List of rotors whose gen_torque will be output (-) [1 to NRotOuts] [unused for NRotOuts=0]
8                     NFlpOuts          !  Number of Flaps whose deflections will be output [0 to 8]
1 2 3  4 5 6 7 8           FlapOuts     !  List of flaps whose deflections will be output (-) [1 to NFlpOuts] [unused for NFlpOuts=0]
OutList          The next line(s) contains a list of output parameters.  See OutListParameters.xlsx for a listing of available output channels (quoted string)
"Rot1GTq,Rot2GTq,Rot3GTq,Rot4GTq,Rot5GTq,Rot6GTq,Rot7GTq,Rot8GTq"
"Rot1Spd,Rot2Spd,Rot3Spd,Rot4Spd,Rot5Spd,Rot6Spd,Rot7Spd,Rot8Spd"
"Flp1Def, Flp2Def, Flp3Def, Flp4Def, Flp5Def, Flp6Def, Flp7Def, Flp8Def"
"DCMG2Bc1, DCMG2Bc2, DCMG2Bc3, DCMG2Bc4, DCMG2Bc5, DCMG2Bc6, DCMG2Bc7, DCMG2Bc8, DCMG2Bc9"
"TethFxb, TethFyb, TethFzb"
END of input file (the word "END" must appear in the first 3 columns of this last OutList line)
```

APPENDIX B: DUMMY-CONTROLLER DATA FILE

Notes:

1. The lines are wrapped due to page limitations;

2. Only the first two lines (two time-steps) of a generic data file are shown below;

3. The file contains two comment lines and one header line before the actual data lines.

4. The File contains all the commanded outputs from the controller: Generation torques, rotor speeds, rotor accelerations, rotor blade pitches in the Makani CSIM rotor/motor ordering and sign conventions, and the flap deflections in the Makani CSIM conventions.

```
#This file contains CTRL output data to be read in by KiteFastController.f90 from data in /home/rdamiani/makani/logs/last.h5

#This file was generated by KFASTvsCSIM on 2020-03-03 13:21:16.260834

#          t[s]    GenTorq1 [Nm]    GenTorq2 [Nm]    GenTorq3 [Nm]    GenTorq4 [Nm]    GenTorq5 [Nm]    GenTorq6 [Nm]    GenTorq7 [Nm]    GenTorq8 [Nm]   RtrSpd1 [rad/s]   RtrSpd2 [rad/s]   RtrSpd3 [rad/s]   RtrSpd4 [rad/s]   RtrSpd5 [rad/s]   RtrSpd6
[rad/s]   RtrSpd7 [rad/s]   RtrSpd8 [rad/s] RtrAcc1 [rad/s^2] RtrAcc2 [rad/s^2] RtrAcc3 [rad/s^2] RtrAcc4 [rad/s^2] RtrAcc5 [rad/s^2] RtrAcc6 [rad/s^2] RtrAcc7 [rad/s^2] RtrAcc8 [rad/s^2]    BlPitch1 [rad]    BlPitch2 [rad]    BlPitch3 [rad]    BlPitch4 [rad]    BlPitch5
[rad]   BlPitch6 [rad]    BlPitch7 [rad]    BlPitch8 [rad]      Flap1 [rad]      Flap2 [rad]      Flap3 [rad]      Flap4 [rad]      Flap5 [rad]      Flap6 [rad]      Flap7 [rad]      Flap8 [rad]

    0.0000000000   -30.1734238280    21.6971167705    12.6404257041    -3.3710331088    61.9785645136   -68.9937414231   -76.1381149685    83.0869760200  -147.0403049350   144.9570916166   142.8475678868  -140.6912989073   164.7884093240  -166.6700659173
-168.5173342090   170.3235903221     7.5206073242    -8.3316864745    -9.2336392530    10.0854291257    -9.4734184840     8.8071522302     8.2872443074    -7.7057721710     0.0000000000     0.0000000000     0.0000000000     0.0000000000     0.0000000000
0.0000000000     0.0000000000     0.0000000000    -0.1374410634    -0.1380903594    -0.0023591218    -0.0023591218    -0.1531278696    -0.1524666892     0.0641672312     0.0146725181

    0.0200000000   -27.8303845244    20.3852262000    12.3413899572    -4.1314297960    62.0855545054   -68.1587733303   -74.4233682894    80.4956624820  -146.8693708672   144.7818510875   142.6656402049  -140.5047064833   164.6091511642  -166.4941887023
-168.3417021134   170.1502927051     8.2944629083    -8.4590640863    -8.7832380358     8.9734402854    -8.8165900979     8.6559515535     8.6766887255    -8.5645287955     0.0000000000     0.0000000000     0.0000000000     0.0000000000     0.0000000000
0.0000000000     0.0000000000     0.0000000000    -0.1371522624    -0.1378012129    -0.0023562864    -0.0023562864    -0.1534118768    -0.1527513647     0.0642831687     0.0150681305
```

APPENDIX C: KITEFASTCONTROLLER_DRIVER.F90 INPUT FILE

Notes:

1. The file follows KiteFAST's convention for the input files.

```
--- KiteFastControllerDriver INPUT FILE i.e. InitInp%DrvInputFile---
Testing Controller launch like this: ~/sandbox/build/modules/kitefast-controller/kitefastcontroller_driver RRD_CTRLDRIVER_input.dat
--- SIMULATION CONTROL ---
False     Echo      ! Echo input data to <RootName>.ech? (flag)
3         numFlaps  ! Number of Flaps per wing; note for Makani's CTRL this must be =3
2         numPylons = 2 ! Number of Pylons per wing; note for Makani's CTRL this must be =2
1.61  1.61  1.61  1.61    SPyRtrIrot(i,j) !Starboard rotor inertias with KFAST order of: do j = 1, p%numPylons [ do i = top,bot]
1.61  1.61  1.61  1.61    PPyRtrIrot(i,j) !Port rotor inertias with KFAST order of: do j = 1, p%numPylons [ do i = top,bot]
1         KFCmode   ! how to calculate control outputs: 0=dummy (hardwired in KiteFASTcontroller.f90), 1=ctrl proper,2=ctrl data file
--- INPUTS ---
'/home/rdamiani/sandbox/glue-codes/kitefast/test_cases/RRD_m600_land_hiStart/KFCdriver_test'        OutFileRoot  ! Root for outputfiles
'/home/rdamiani/sandbox/glue-codes/kitefast/test_cases/RRD_m600_land_hiStart/RRD_ctrl_input.dat'     InputFileName  !the input file for the controller routine
1         KFCdrivermode ! 1=reading inputs for the ctrl from this file; 2=old static test of the controller lib with one step only and hardcoded parameters
'/home/rdamiani/sandbox/glue-codes/kitefast/test_cases/RRD_m600_land_hiStart/CSIMstates.inp'      DrvCtrlInputs  !the data file containing the time series of CTRL inputs,
ignored if  KFCdrivermode=2

END of input file (the word "END" must appear in the first 3 columns of this last line)
```

APPENDIX D: KFC DRIVER STATES DATA FILE

Notes:

1. The lines are wrapped due to page limitations;

2. Only the first two lines (two time-steps) of a generic states data file are shown below;

3. The file contains two comment lines and one header line before the actual data lines.

4. The File contains all the states that the KFC API passes to STIctrl: time stamp, air density, Xg, Vg, Vb, Ag, Ab, Acc_norm, pqr, , wind_g, apparent_wind, tether_forceb_c , dcm_g2b, aeroTorq.

5. The states are in the Makani CSIM rotor/motor ordering and sign conventions, except for those that are being manipulated by STICtrl (convention was not enforced) such as apparent wind and tether force which are given as vectors in the kite CS (not kite's spherical CS).

```
#This file (ignored if  KFCdrivermode=2) contains CTRL input data to be read in by KFCdriver from data in /home/rdamiani/makani/logs/last.h5

#This file was generated by KFASTvsCSIM on 2020-03-26 16:38:08.756679

#        t_c        rho_c        Xg_c_x        Xg_c_y        Xg_c_z        Vg_c_x        Vg_c_y        Vg_c_z        Ag_c_x        Ag_c_y        Ag_c_z     acc_norm_c        Vb_c_x        Vb_c_y        Vb_c_z        Ab_c_x        Ab_c_y        Ab_c_z
pqr_c(1)     pqr_c(2)     pqr_c(3)   wind_g_c(1)   wind_g_c(2)   wind_g_c(3)  app_wind_c(1)  app_wind_c(2)  app_wind_c(3)  tether_vec_c(1)  tether_vec_c(2)  tether_vec_c(3)   dcm_g2b_c(1)   dcm_g2b_c(2)   dcm_g2b_c(3)   dcm_g2b_c(4)
dcm_g2b_c(5)   dcm_g2b_c(6)   dcm_g2b_c(7)   dcm_g2b_c(8)   dcm_g2b_c(9)    aeroTorq(1)    aeroTorq(2)    aeroTorq(3)    aeroTorq(4)    aeroTorq(5)    aeroTorq(6)    aeroTorq(7)    aeroTorq(8)

#        [s]     [kg/m3]        [m]        [m]        [m]       [m/s]       [m/s]       [m/s]      [m/s2]      [m/s2]      [m/s2]      [m/s2]       [m/s]       [m/s]       [m/s]      [m/s2]      [m/s2]      [m/s2]     [rad/s]     [rad/s]
[rad/s]       [m/s]       [m/s]       [m/s]       [m/s]       [m/s]       [m/s]        [N]        [N]        [N]        [-]        [-]        [-]        [-]        [-]        [-]        [-]        [-]        [-]       [Nm]       [Nm]       [Nm]
[Nm]        [Nm]        [Nm]        [Nm]        [Nm]

    0.0000000000    1.0260000000  -299.6791918037    17.5337634440  -313.7610198871    10.5183018943   -33.0589012385   -11.8170092933    -3.9615052096    -4.5172864521     7.6946540470     6.8658938740    35.2535765754    -7.6081817280    -6.5167663688   -
1.1539234932    -9.4391120086     2.2087839144    -0.0449720215    -0.0606116668    -0.2278568417   -12.0000000000     0.0000000000     0.0000000000  1110.0079401506     1.4635830597    41.6535870406 -6121.4468875130 -6009.9265846743 54336.9427352959
0.5333342054     0.5120498890     0.6733197877    -0.7779822245     0.6094287257     0.1527752816    -0.3321118542    -0.6053111097     0.7233948969    41.9055712538   -34.6945476707   -27.0449029388    19.1043025449   -76.7570973486    82.7328989022
89.0662160880   -95.1079806068

    0.0200000000    1.0260000000  -299.4696188411    16.8716836891  -313.9958199042    10.4389544241   -33.1489883319   -11.6629316652    -3.9734573131    -4.4915472684     7.7128905045     6.8658938740    35.2574696291    -7.6304096341    -6.5221818507   -
1.1406375109    -9.4478941412     2.2107198340    -0.0455519324    -0.0605491441    -0.2280677096   -12.0000000000     0.0000000000     0.0000000000  1114.6508483502     1.4639876600    41.6392036782 -6118.6013207932 -5995.0844030431 54258.8451459152
0.5318111708     0.5138684978     0.6731389497    -0.7805663483     0.6057368107     0.1542695436    -0.3284707819    -0.6074718784     0.7232460593    40.7697466614   -33.5813661746   -26.0432412930    18.1299966412   -75.8394350581    81.6620577537
87.9590027011   -93.8563274030
```